



Efinity[®] Debugger Tutorial

UG-EFN-TUTDBG-v1.4
March 2025
www.efinixinc.com



Contents

Introduction.....	3
Tutorial: Automated Debugging Flow.....	3
Prepare the Tutorial Files.....	3
Create a Debug Profile.....	4
Program the T20 FPGA.....	4
Run the Debugger.....	5
Tutorial: Manual Debugging Flow.....	5
Prepare the Tutorial Files.....	5
Create a Debug Profile.....	6
Add Debug Code to Your Project.....	7
Debugging with Multiple Manual Debug Profiles.....	9
Program the T20 FPGA.....	10
Run the Debugger.....	11
Where to Learn More.....	12
Revision History.....	12

Introduction

The Efinity® software includes a hardware Debugger to probe signals in your FPGA design via the JTAG interface. The Debugger includes two debug cores:

- You use a manual flow and the Profile Editor to configure Virtual I/O (vio) cores.
- You can use a manual flow or the Debug Wizard's automated flow to configure Logic Analyzer (la) cores.

The following sections walk you through the Debugger's automated and manual flows.



Note: The Debugger tutorials require the Trion® T20 BGA256 Development Board and Efinity® software v2019.3 or higher. These tutorials assume that you have already installed the Efinity® software and USB driver for the board.

Tutorial: Automated Debugging Flow

This tutorial walks you through the Debugger automated flow using an example **helloworld** design. This tutorial uses the Trion® T20 BGA256 Development Board, the GTKWave waveform viewer, and assumes that you have working knowledge of the Efinity® software.

You add a Logic Analyzer debug core and configure it using the Debug Wizard.

Prepare the Tutorial Files

In this step you set up your environment and copy the Debugger tutorial design to your working directory.

1. Run the Efinity setup script if you have not already done so:
 - Linux: `source <Efinity path>/bin/setup.sh`
 - Windows: `<Efinity path>\bin\setup.bat`
2. Copy the folder `<Efinity path>/debugger/demo/helloworld-dbg` to your working directory.
3. Connect the Trion® T20 BGA256 Development Board to your computer using a USB cable.

[Create a Debug Profile >](#)

Create a Debug Profile

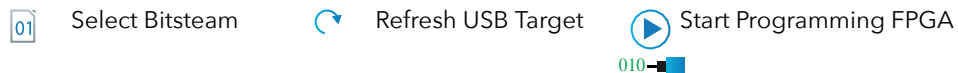
In this task you add the Logic Analyzer debug core and configure it.

1. Open the **helloworld** project in the **helloworld-dbg** directory.
2. Synthesize the design. You do not need to do a full compile; the Debug Wizard only uses the post-map netlist.
3. Click the Debug Wizard icon in the main icon bar to launch it.
4. In the **Signals from** list, choose **Elaborated Netlist** to browse for signals in the pre-map netlist, or **Post-Map** to use signals from the post-map netlist.
5. Select the `led` and `counter` buses from the list on the left and use the **>>** button to move them to the right. Leave the **Probe Type** at the default, which is **DATA AND TRIGGER**.
6. Click **Next**. The wizard generates a debug profile.
7. Leave **Enable "Auto Instantiation"** turned on. This option enables the debug profile in your project. Click **Finish**.
8. The software prompts you to recompile. Click **OK**.
9. Perform a full compile.

< [Prepare the Tutorial Files](#) on page 3

[Program the T20 FPGA](#) on page 4 >

Program the T20 FPGA



You program the Trion® T20 FPGA on the development board using these steps:

1. Choose **Tools > Open Debugger** to launch the Debugger. The programming controls are in the Program box.
2. The Trion T20 Development Board displays as the **USB Target**. If it does not, make sure that the board is connected to your computer and click **Refresh USB Targets**.
3. Click the **Select Image File** button.
4. Browse to the **outflow** directory and choose `<helloworld>.bit`.
5. Click **Start Programming**. The console displays programming messages.

< [Program the T20 FPGA](#) on page 4

[Run the Debugger](#) on page 5 >

Run the Debugger



Connect Debugger



Disconnect Debugger



Add Net

After you program the FPGA with the design containing the debug core, you can run the Debugger to observe the values on the probed signals. In the Debugger:

1. Click Connect Debugger.
2. In the **Trigger Setup** tab, click Add Net.
3. Select **led[7:0]** and click **OK**.
4. Specify a **Value** of 00001111, which triggers when the LED output is 00001111.
5. Click **Run**. The Debugger waits for the trigger and then captures data.
6. When the Debugger finishes, it automatically opens the waveform in GTKWave
7. Click Disconnect Debugger to stop the Debugger.

< [Program the T20 FPGA](#) on page 10

Tutorial: Manual Debugging Flow

This tutorial walks you through the Debugger manual flow using an example **helloworld** design. This tutorial uses the Trion® T20 BGA256 Development Board, the GTKWave waveform viewer, and assumes that you have working knowledge of the Efinity® software.

You add Logic Analyzer and Virtual I/O cores manually in the Debugger's Profile Editor perspective.



Note: If you are switching from automated debugging flow to manual debugging flow, you need to go to **File > Edit Project > Debugger(tab)** and deselect **Debugger Auto Instantiation**.

Prepare the Tutorial Files

In this step you set up your environment and copy the Debugger tutorial design to your working directory.

1. Run the Efinity setup script if you have not already done so:
 - Linux: `source <Efinity path>/bin/setup.sh`
 - Windows: `<Efinity path>\bin\setup.bat`
2. If you have not already done so, copy the folder `<Efinity path>/debugger/demo/helloworld-dbg` to your working directory.
3. Connect the Trion® T20 BGA256 Development Board to your computer using a USB cable.

[Create a Debug Profile](#) on page 6 >

Create a Debug Profile



Add Debug Core



Add Probe



Add Source

In this task you add Virtual I/O and Logic Analyzer debug cores to a profile and configure them:

- You add the input signal(s) to control and the output signal(s) to observe to the Virtual I/O core.
- You add the wire, register, or signal to observe to Logic Analyzer core.

Remember to map the clock source after you instantiate the debug core.

1. Open the **helloworld** project in the **helloworld-dbg** directory.
2. Choose **Tools > Open Debugger** to launch the Debugger. Because your project does not have a debug profile, the Debugger opens to the Profile Editor perspective.
3. Click **Add Debug Core > Virtual I/O** to add a new core with the default **Core name** (**vio0**).
4. Add three probes (in) and three sources (out) with the following name and width settings (leave the other settings at the defaults):

Name	Type	Width	Description
counter	Probe	26	Shows the values for counter[25:0].
raddr	Probe	4	Shows the values for raddr[3:0].
led	Probe	8	Shows the values for the pattern on led[7:0].
vio_reverse	Source	1	Control the reverse button.
vio_mux_sel	Source	1	Control the multiplexer select.
vio_maddr	Source	4	Control the memory address maddr[3:0].

5. Click **Add Debug Core > Logic Analyzer** to add a second core with the default **Core name** (**la0**).
6. Add three probes (in) with the same name and width settings as the probes in the VIO core (leave the other settings at the defaults):

Name	Type	Width	Description
counter	Probe	26	Captures values in counter[25:0]
raddr	Probe	4	Captures the values in maddr[3:0]
led	Probe	8	Captures the values in led[7:0]

7. Click **Generate Debug RTL**. The Debugger creates the file **debug_top.v** and template files (**debug_TEMPLATE.v** and **debug_TEMPLATE.vhd**) in your project directory.
8. Open the **debug_top.v** file and rename **edb_top** as **edb_top_manual**.

9. Close the Debugger.

< [Prepare the Tutorial Files](#) on page 5[Add Debug Code to Your Project](#) on page 7 >

Add Debug Code to Your Project

When you generate the debug code, the software copies the **debug_top.v** file to your project directory. You need to add the file to your project, instantiate the RTL, and compile.

1. In the Efinity® main window, click the Project tab under the dashboard.
2. Right-click **Design** and choose **Add**.
3. Browse to your project directory.
4. Select the **debug_top.v** and click **Open**.
5. Add the JTAG User Tap block to the interface design.
 - a) Open the Interface Designer.
 - b) Select **JTAG User Tap**.
 - c) Click Add Block.
 - d) Choose **JTAG_USER1** as the **JTAG Resource**.
 - e) Generate SDC constraints.
 - f) Close the Interface Designer.
6. Edit the **helloworld.v** design to enable the debug code:
 - a) Add all of the JTAG input and output pins to the project top module (**helloworld**). Refer to the JTAG User TAP block pin names in the Interface Designer for a full list of pin names; alternatively, you can paste the following into the **helloworld** IO port module:

```
input jtag_inst1_CAPTURE,
input jtag_inst1_DRCK,
input jtag_inst1_RESET,
input jtag_inst1_RUNTEST,
input jtag_inst1_SEL,
input jtag_inst1_SHIFT,
input jtag_inst1_TCK,
input jtag_inst1_TDI,
input jtag_inst1_TMS,
input jtag_inst1_UPDATE,
output jtag_inst1_TDO,
```

- b) Declare the following probe signal and width in the **helloworld** module:

```
wire vio_reverse;
wire vio_mux_sel;
wire [AWIDTH-1:0] vio_maddr;
```

- c) Change hardware pushbutton to the **vio0** source

Old	New
<pre>assign raddr = counter[COUNTER_SIZE-1:DELAY_SIZE];</pre>	<pre>assign raddr = (vio_mux_sel)? vio_maddr : counter[COUNTER_SIZE-1:DELAY_SIZE];</pre>

- d) Change hardware pushbutton to the **vio0** source

Old	New
<pre>else if (~reverse) begin</pre>	<pre>else if (~reverse vio_reverse) begin</pre>

- e) Instantiate the debug core in the project's top module. You can copy the example code from the generated `debug_TEMPLATE.v` or `debug_TEMPLATE.vhd` file in the project folder and rename the module `edb_top` to `edb_top_manual` or copy the following code into the project's top module.

```
edb_top_manual edb_top_inst_manual (
    .bscan_CAPTURE ( jtag_inst1_CAPTURE ),
    .bscan_DRCK ( jtag_inst1_DRCK ),
    .bscan_RESET ( jtag_inst1_RESET ),
    .bscan_RUNTEST ( jtag_inst1_RUNTEST ),
    .bscan_SEL ( jtag_inst1_SEL ),
    .bscan_SHIFT ( jtag_inst1_SHIFT ),
    .bscan_TCK ( jtag_inst1_TCK ),
    .bscan_TDI ( jtag_inst1_TDI ),
    .bscan_TMS ( jtag_inst1_TMS ),
    .bscan_UPDATE ( jtag_inst1_UPDATE ),
    .bscan_TDO ( jtag_inst1_TDO ),
    .vio0_clk ( clk ),
    .vio0_counter ( counter ),
    .vio0_raddr ( raddr ),
    .vio0_led ( led ),
    .vio0_vio_reverse ( vio_reverse ),
    .vio0_vio_mux_sel ( vio_mux_sel ),
    .vio0_vio_maddr ( vio_maddr ),
    .la0_clk ( clk ),
    .la0_counter ( counter ),
    .la0_raddr ( raddr ),
    .la0_led ( led )
);
```

- f) Save the `helloworld.v`

7. Compile the design.



Note: For advice on using an automated flow for adding a logic analyzer core, refer to the "Debug Wizard" in the **Efinity Software User Guide** Efinity Software User Guide.



Note: The project in `<Efinity path>/debugger/demo/helloworld-dbg_GOLDEN` has performed steps 1-7 with the original helloworld project. This project is used for the BRAM Initial Content Updater (see: "About the BRAM Initial Content Updater" in **Efinity Software User Guide** Efinity Software User Guide).

< [Create a Debug Profile](#) on page 6

[Debugging with Multiple Manual Debug Profiles](#) on page 9 >

Debugging with Multiple Manual Debug Profiles



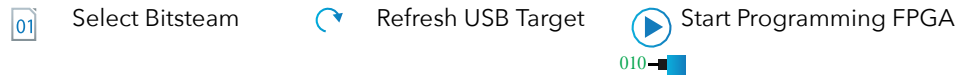
You can use multiple debug profiles within the same project. Each debugger profile must use a different set of JTAG User Tap resources, and you must enable the profile in the Interface Designer. The **helloworld-dbg_MULTI_USERS** uses JTAG User TAP 1 for the Virtual I/O debugger profile (**vio_debug_profile.json**) and JTAG User TAP 2 for the Logic Analyzer debugger profile (**la_debug_profile.json**).

1. Open the **helloworld** project in the *<Efinity install path> /debugger/demo.helloworld-dbg_MULTI_USERS* directory.
2. Compile the project.
3. Configure the FPGA using the debugger, JTAG mode, and the **.bit** file located in the project's outflow directory.
4. Debug with the Logic Analyzer debugger profile:
 - a) Change to **Perspectives > Profile Editor**.
 - b) Click **Import Profile**.
 - c) Choose the file **la_debug_profile.json** and click **OK**.
 - d) Change to the Debugger using **Perspectives > Debug**.
 - e) Change the User TAP to **USER2**.
 - f) Connect the debugger.
 - g) Disconnect the debugger when finished.
5. Debug with the Virtual I/O debugger profile:
 - a) Change to **Perspectives > Profile Editor**.
 - b) Click **Import Profile**.
 - c) Choose the file **vio_debug_profile.json** and click **OK**.
 - d) Change to the Debugger using **Perspectives > Debug**.
 - e) Change the User TAP to **USER1**.
 - f) Connect the debugger.
 - g) Disconnect the debugger when finished.

< [Add Debug Code to Your Project](#) on page 7

[Program the T20 FPGA](#) on page 10 >

Program the T20 FPGA






You program the Trion® T20 FPGA on the development board using these steps:

1. Choose **Tools > Open Debugger** to launch the Debugger. The programming controls are in the Program box.
2. The Trion T20 Development Board displays as the **USB Target**. If it does not, make sure that the board is connected to your computer and click **Refresh USB Targets**.
3. Click the **Select Image File** button.
4. Browse to the **outflow** directory and choose *<helloworld>.bit*.
5. Click **Start Programming**. The console displays programming messages.

< [Debugging with Multiple Manual Debug Profiles](#) on
page 9

[Run the Debugger](#) on page 11 >

Run the Debugger

 Connect Debugger
  Disconnect Debugger
  Add Trigger Condition

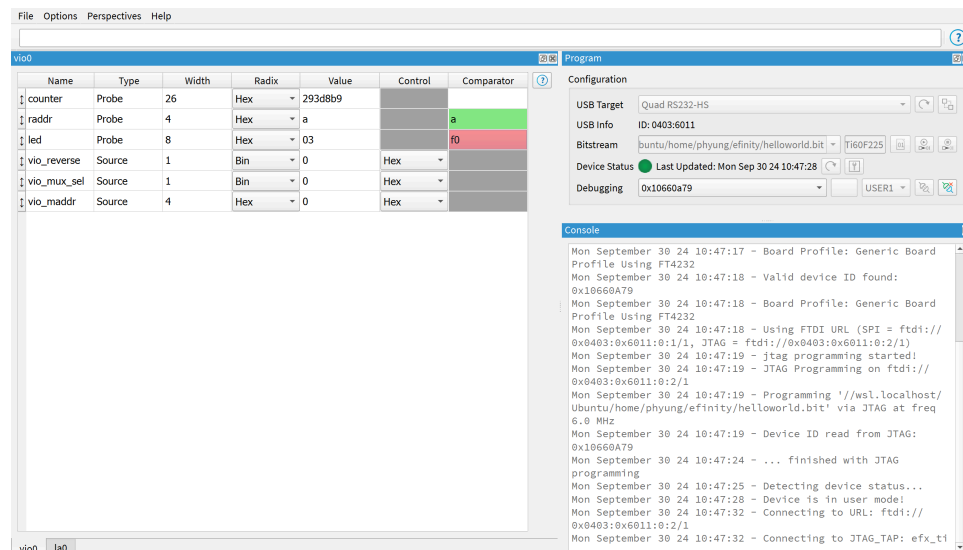
After you program the FPGA with the design containing the debug core, you can run the Debugger to observe the values on the probed signals. In the Debugger:

1. Click Connect Debugger. The view opens to the **la0** tab.
2. In the **Trigger Setup** tab, click Add Trigger Condition.
3. Choose **led[7:0]** and click **OK**.
4. Specify a value of **00001111**, which triggers when the LED output is 00001111.
5. Click the **vio0** tab. The **Value** fields show the data captured on the probes.
To reverse the LED blinking direction, change the **Value** for **vio_reverse** to **1** and press Enter. To stop the LEDs blinking, change the **Value** for **vio_mux_sel** to **1** and press Enter.
6. Click Disconnect Debugger to stop capturing data.



Note: The **Comparator** is a GUI-only feature. When the probe value is equal to the compared value, the cell background turns green to indicate a match. If the values do not match, the cell background turns red. By default, the compared values are an empty string, with the cell background color not set.

Figure 1: Example of Debugger with Comparator in action



Note: Debug profiles are paired to their respective generated debug core via a UUID key. Ensure that the correct debug profile is selected when connecting to the debugger.

< [Program the T20 FPGA](#) on page 10

Where to Learn More

The Efinity® software includes documentation as PDF user guides and on-line HTML help. This documentation is provided with the software. You can also access the latest versions of PDF documentation in the Support Center:

- [Efinity Software User Guide](#) Efinity Software User Guide
- [Efinity Synthesis User Guide](#) Efinity Synthesis User Guide
- [Efinity Timing Closure User Guide](#) Efinity Timing Closure User Guide
- [Efinity Software Installation User Guide](#) Efinity Software Installation User Guide
- [Efinity Trion Tutorial](#) Efinity Trion Tutorial
- [Efinity Debugger Tutorial](#) Efinity Debugger Tutorial
- [TitaniumTJ-Series Interfaces User Guide](#) Titanium Interfaces User Guide
- [Trion Interfaces User Guide](#) Trion Interfaces User Guide
- [Efinity Interface Designer Python API](#) Efinity Interface Designer Python API
- [Quantum® Trion Primitives User Guide](#) Quantum® Trion Primitives User Guide
- [Quantum® TitaniumTJ-Series Primitives User Guide](#) Quantum® Titanium Primitives User Guide
- [Quantum® TopazTP-Series Primitives User Guide](#) Quantum® Primitives User Guide

In addition to documentation, Efinix field application engineers have created a series of videos to help you learn about aspects of the software. You can view these videos in the Support Center.

Revision History

Table 1: Revision History

Date	Version	Description
March 2025	1.4	Notes added to Tutorial: Manual Debugging Flow on page 5 and Run the Debugger on page 11. (DOC-2232)
November 2024	1.3	Added paragraph explaining the functionality of the Comparator, as well as an example illustration. (DOC-2153)
September 2024	1.2	Major revision of Add Debug Code to Your Project on page 7. (DOC-1848) Added new section: Debugging with Multiple Manual Debug Profiles on page 9. (DOC-1848) Rearranged the tables in topic Create a Debug Profile on page 6.
December 2022	1.1	Corrected the source and probe names for the manual flow's Run the Debugger topic.
August 2022	1.0	Initial release. The content in this tutorial originally appeared in the Efinity Trion Tutorial.